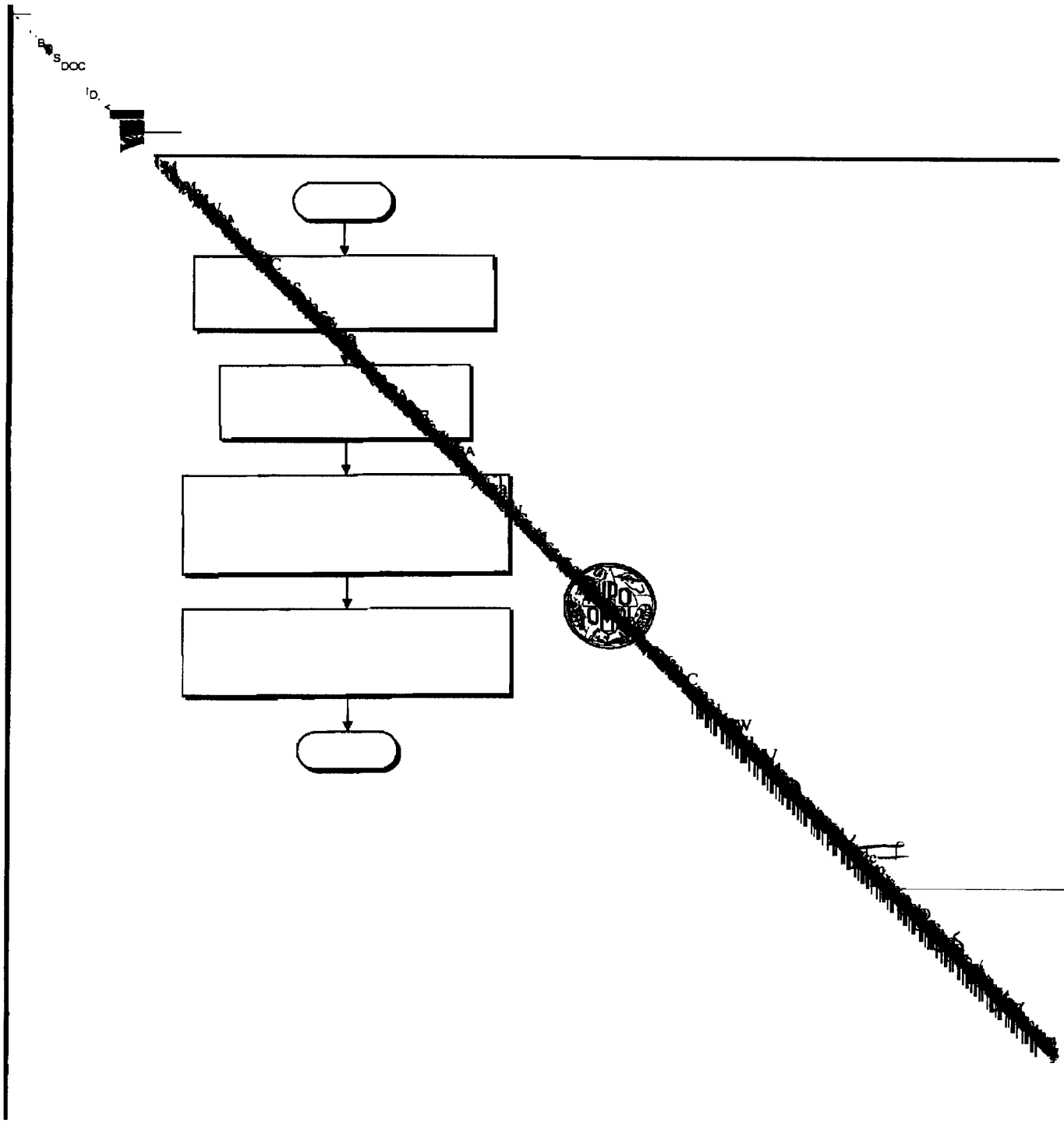


Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND APPARATUS FOR USING AN ASSIST PROCESSOR TO PREFETCH INSTRUCTIONS FOR A PRIMARY PROCESSOR

Inventor(s): Shailender Chaudhry and Marc Tremblay

BACKGROUND

Field of the Invention

The present invention relates to techniques for improving computer system performance. More specifically, the present invention relates to a method and an apparatus for prefetching instructions from memory by using an assist processor to perform prefetch operations in advance of a primary processor.

Related Art

As increasing semiconductor integration densities allow more transistors to be integrated onto a microprocessor chip, computer designers are investigating different methods of using these transistors to increase computer system performance. To this end, computer designers are beginning to incorporate multiple central processing units (CPUs) into a single semiconductor chip. This can result in performance gains for computational tasks that can be parallelized (divided) into separate pieces that can be concurrently executed.

Unfortunately, performance gains from parallelization can be limited for many applications that contain inherently serial portions of code. For these inherently serial portions of code, performance is further limited by memory latency problems.

Memory latency problems are growing progressively worse as processor clock speeds continue to improve at an exponential rate. At today's processor clock speeds, it can take as many as 200 processor clock cycles to pull a cache line in from main memory.

Computer designers presently use a number of techniques to decrease memory latency delays. (1) Out-of-order execution can be used to schedule loads and stores so

that memory latency is hidden as much as possible. Unfortunately, out-of-order execution is typically limited to hiding a few clock cycles of memory latency. (2) A non-faulting load instruction can be used to speculatively load a data value without causing a fault when the address is not valid. (3) A steering load instruction can be used to speculatively load a data value into L2 cache, but not L1 cache, so that L1 cache is not polluted by unused data values. Unfortunately, using non-faulting loads and steering loads can result in unnecessary loads. This wastes instruction cache space and ties up registers. (4) Some researchers have investigated using hardware prefetch engines, but these hardware prefetch engines are typically ineffective for irregular memory access patterns.

Memory latency delays can also be a problem during instruction fetch operations. Note that an instruction cache miss can cause as much of a delay as a data cache miss. Also note that it is very hard to predict which instructions are likely to be executed next because of the numerous branches and function calls that are commonly interspersed into program code written in modern programming languages.

What is needed is a method and an apparatus that reduces memory latency delays during instruction fetch operations.

SUMMARY

One embodiment of the present invention provides a system that prefetches instructions by using an assist processor to perform prefetch operations in advance of a primary processor. The system operates by executing executable code on the primary processor, and simultaneously executing a reduced version of the executable code on the assist processor. This reduced version of the executable code executes more quickly than the executable code, and performs prefetch operations for the primary processor in advance of when the primary processor requires the instructions. The system also stores the prefetched instructions into a cache that is accessible by the primary processor so that the primary processor is able to access the prefetched instructions without having to retrieve the prefetched instructions from a main memory.

In one embodiment of the present invention, prior to executing the executable code, the system compiles source code into executable code for the primary processor.

Next, the system profiles the executable code to create instruction traces for frequently referenced portions of the executable code. The system then produces the reduced version of the executable code for the assist processor by producing prefetch instructions to prefetch portions of the instruction traces into a cache that is accessible by the primary processor. The system also inserts communication instructions into the executable code for the primary processor and into the reduced version of the executable code for the assist processor to transfer progress information from the primary processor to the assist processor. This progress information triggers the assist processor to perform the prefetch operations.

In one embodiment of the present invention, the process of compiling the source code and the process of producing the reduced version of the executable code are carried out by a compiler.

In one embodiment of the present invention, if the progress information indicates to the assist processor that the assist processor has prefetched instructions down the wrong path, the reduced version of the executable code causes the assist processor to discontinue prefetching.

In one embodiment of the present invention, the reduced version of the executable code is configured to read control flow history information from special-purpose hardware that records branch history information and call history information. Next, the reduced version of the executable code constructs a predicted path through the executable code based on the control flow history information, and then performs prefetch operations down the predicted path in order to prefetch instructions for the primary processor.

In one embodiment of the present invention, producing the reduced version of the executable code involves constructing a control flow graph for the executable code. In doing so, the system removes loops from the control flow graph, and removes executable code instructions unrelated to the control flow graph. The system also inserts the prefetch instructions into the reduced version of the executable code to prefetch instructions from the executable code for the primary processor.

In one embodiment of the present invention, performing the prefetch operations involves prefetching cache blocks containing multiple instructions for the primary processor.

In one embodiment of the present invention, the system periodically sends the progress information from the primary processor to the assist processor through a one-way communication channel.

In one embodiment of the present invention, the primary processor and the assist processor reside on the same semiconductor chip.

In one embodiment of the present invention, the primary processor and the assist processor reside on distinct semiconductor chips.

In one embodiment of the present invention, the assist processor is a simplified version of the primary processor.

BRIEF DESCRIPTION OF THE FIGURES

FIG. 1 illustrates a computer system in accordance with an embodiment of the present invention.

FIG. 2 illustrates the compilation process in accordance with an embodiment of the present invention.

FIG. 3 is a flow chart illustrating the process of generating the reduced version of the executable code in accordance with an embodiment of the present invention.

FIG. 4 illustrates an example structure for the reduced executable code in accordance with an embodiment of the present invention.

FIG. 5 is a flow chart illustrating details of how to construct the reduced version of the executable code in accordance with an embodiment of the present invention.

FIG. 6 is a flow chart illustrating how the reduced version of the executable code bails out when it determines that it has proceeded down the wrong path in accordance with an embodiment of the present invention.

FIG. 7 is a flow chart illustrating how the reduced version of the executable code operates by reading control flow history information from special-purpose hardware in accordance with an embodiment of the present invention.

FIG. 8 illustrates special-purpose hardware for storing control flow history information in accordance with an embodiment of the present invention.

FIG. 9 illustrates a sample portion of reduced executable code in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION

The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. Various modifications to the disclosed embodiments will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the present invention. Thus, the present invention is not intended to be limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

The data structures and code described in this detailed description are typically stored on a computer readable storage medium, which may be any device or medium that can store code and/or data for use by a computer system. This includes, but is not limited to, magnetic and optical storage devices such as disk drives, magnetic tape, CDs (compact discs) and DVDs (digital video discs), and computer instruction signals embodied in a transmission medium (with or without a carrier wave upon which the signals are modulated). For example, the transmission medium may include a communications network, such as the Internet.

Computer System

FIG. 1 illustrates a computer system 101 in accordance with an embodiment of the present invention. Computer system 101 includes a primary processor 102 and an assist processor 104. All of these structures reside on silicon die 100 (although in other embodiments of the present invention they can reside on multiple silicon dies).

Processors 102 and 104 include instruction caches 112 and 120, respectively, which contain instructions to be executed by processors 102 and 104.

Processors 102 and 104 additionally include load buffers 114 and 122 as well as store buffers 116 and 124 for buffering communications with data caches 107 and 106, respectively. More specifically, primary processor 102 includes load buffer 114 for buffering loads received from data cache 107, and store buffer 116 for buffering stores to data cache 107. Similarly, assist processor 104 includes load buffer 122 for

buffering loads received from data cache 106, and store buffer 124 for buffering stores to data cache 106.

Processors 102 and 104 are additionally coupled together by one-way communication channels 128-129, which facilitates rapid communication between primary processor 102 and assist processor 104. Note that communication channel 128 allows primary processor 102 to write into register 126 within assist processor 104. Similarly, communication channel 129 allows assist processor 104 to write into register 127 within primary processor 102.

Unlike using shared memory communication mechanisms, writing into register 126 (or 127) will not cause a cache miss or coherence traffic. Furthermore, primary processor 102 does not have to wait until assist processor 104 receives the communication to continue processing. In contrast, if primary processor 102 were to communicate with assist processor 104 through memory, the system would have to wait for store buffer 116 to be cleared in order to communicate.

In one embodiment of the present invention, data caches 106 and 107 are 16K-byte 4-way set-associative data caches with 32-byte cache lines.

Data cache 106, data cache 107, instruction cache 112 and instruction cache 120 are coupled through switch 110 to L2 cache 113. Switch 110 may include any type of circuitry for switching signal lines. In one embodiment of the present invention, switch 110 is a cross bar switch.

L2 cache is a large unified cache for storing both instructions and data for primary processor 102 and assist processor 104. L2 cache 113 is coupled to memory controller 111, which is itself coupled to dynamic random access memory (DRAM) 108 (located off chip).

DRAM 108 contains executable code 130 for primary processor 102. During system operation, executable code 130 is loaded through memory controller 111, L2 cache 113 and switch 110 into instruction cache 112 of primary processor 102. DRAM 108 also contains reduced executable code 132 for assist processor 104. Reduced executable code 132 is a reduced version of executable code 130 that generates the same pattern of memory references as executable code 130. During system operation, reduced executable code 132 is loaded through memory controller 111, L2 cache 113 and switch 110 into instruction cache 120 of assist processor 104.

DRAM 108 additionally contains data 134 that is moved to and from data caches 106-107 through memory controller 111, L2 cache 113 and switch 110.

Note that the present invention can be used in any computer system that includes multiple processors, and is not limited to the illustrated computer system structure.

Also note that in one embodiment of the present invention, assist processor 104 is used to make instruction accesses in advance of when the instructions are used by primary processor 102. In this embodiment, assist processor 104 does not have to actually perform all of the computations specified in a program; assist processor 104 merely has to perform sufficient computations to allow assist processor 104 to determine the instruction access pattern of the program. Hence, assist processor 104 can be a much simpler version of primary processor 102, excluding circuitry that is not required to perform instruction address computations (e.g., dividers and multiplier circuits).

Compilation Process

FIG. 2 illustrates the compilation process in accordance with an embodiment of the present invention. During the compilation process, source code 202 feeds through compiler 204 to produce executable code 130 for primary processor 102. Executable code 130 then feeds through reduction module 208 to produce reduced executable code 132 for assist processor 104. Note that reduction module 208 may be part of compiler 204, or alternatively, may be separate from compiler 204.

In one embodiment of the present invention, the compilation and reduction processes take place before run-time. In another embodiment, the compilation and reduction processes take place during run-time, while the program is executing.

Generating Reduced Executable Code

FIG. 3 is a flow chart illustrating the process of generating reduced executable code 132 within reduction module 208 in accordance with an embodiment of the present invention. The system starts by profiling (simulating execution of) executable code 130 to identify hot spots in which memory latency is causing delays (step 302). The system uses information gained during the profiling process to build instruction

traces of the hot spots in executable code 130. In one embodiment of the present invention, the system constructs about 100 dispersed traces of about 2000 instructions each. The system then correlates these traces back to the corresponding source code (step 304).

Next, the system creates reduced executable code 132 by producing code to prefetch the traces (step 306). This process is described in more detail below with reference to FIGs. 4, 5 and 9.

The system also inserts processor-to-processor communication code into both executable code 130 and reduced executable code 132 (step 308). This communication code causes primary processor 102 to communicate progress information to assist processor 104. This progress information enables assist processor 104 to determine whether it is prefetching down the correct path and whether it is too far in advance of primary processor 102 to continue prefetching.

Note that it is undesirable for assist processor 104 to prefetch instructions too far in advance of primary processor 102, because instructions that are prefetched far in advance are less likely to be used by the primary processor 102, and may replace data needed more immediately by primary processor 102. Prefetching instructions that are less likely to be used can tie up memory bandwidth and can lead to cache pollution, which reduces computer system performance. Hence, it is desirable for assist processor 104 to wait until the execution path of primary processor 102 is close to the corresponding code in assist processor 104 before initiating prefetch instructions.

Also note that a single prefetch operation typically retrieves an entire cache line containing multiple instructions. For example, a prefetched cache line may include 64 bytes that contain 16 4-byte instructions.

If this progress information indicates that assist processor 104 is prefetching down the wrong path through executable code 130, the system can cause assist processor 104 to bail out of prefetching the instruction trace.

FIG. 4 illustrates one possible structure for reduced executable code 132 in accordance with an embodiment of the present invention. In this embodiment, reduced executable code 132 is structured as a loop. The code first starts by reading a variable "val", which can be located in a shared memory or within register 126 in

FIG. 1. If the val does not equal zero, the system executes a switch statement that executes specific pieces of code containing prefetch operations based upon the value contained in val. In this way, primary processor 102 can communicate progress information to assist processor 104 through the variable val. This progress information causes specific prefetch instructions to be executed so that assist processor 104 prefetches instructions in advance of when primary processor 102 requires the instructions. The code then resets val to zero and continues with the loop. Note that reads and writes to val are not synchronized.

FIG. 5 is a flow chart illustrating details of how to construct the reduced executable code 132 in accordance with an embodiment of the present invention. The system starts by constructing a control flow graph for executable code 130 (step 502). Next, the system removes small loops from the control flow graph (step 504) because small loops are presumed to be prefetched into L2 cache 113. Hence, once a small loop is prefetched, it executes until the loop is finished. The system also removes all instructions from executable code 130 that are unrelated to the control flow graph (step 506). This can be done because the unrelated code will not affect the execution path through the code, and hence will have no influence on the required prefetch operations. For example, deterministic code that performs mathematical computations can be removed.

Next, the system inserts prefetch instructions into reduced executable code 132 to prefetch instructions for corresponding sections of executable code 130 (step 508).

Bail Out Process

FIG. 6 is a flow chart illustrating how reduced executable code 132 bails out when it determines that it has proceeded down the wrong path in accordance with an embodiment of the present invention. The system starts when reduced executable code 132 executing on assist processor 104 reads progress information that was written by primary processor 102 (step 602). This progress information indicates where the execution path of primary processor 102 has gone within executable code 130.

If this progress information indicates that assist processor 104 is proceeding down the wrong path, assist processor 104 discontinues prefetching the trace (step 604). Assist processor may optionally start prefetching for the correct path.

Note that during the prefetching process, assist processor 104 may decide to prefetch only down one side of a branch, and primary processor 102 may actually decide to go down the other side of the branch. In this case, assist processor 104 has taken the wrong path.

Also note that assist processor 104 does not necessarily have to bail out. The only penalty for prefetching down the wrong path is cache pollution and unnecessary memory transfers, which may not be a significant factor for a very small trace.

Special-Purpose Hardware for Storing History Information

FIG. 7 is a flow chart illustrating how reduced executable code 132 operates in accordance with an embodiment of the present invention. The system starts by reading control flow history information from special-purpose hardware that records control flow history information for executable code 130 (step 702). For example, this control flow history information can include a record of which branches are taken and which function calls are executed. Next, the system constructs a predicted path through executable code 130 based upon the control flow history information (step 704). For example, the predicted path can be determined based upon the last path taken through executable code 130, or based upon the most frequently taken path through executable code 130. Next, reduced executable code 132 causes assist processor 104 to perform prefetch operations down the predicted path (step 706).

FIG. 8 illustrates special-purpose hardware for storing control flow history information in accordance with an embodiment of the present invention. In this embodiment, the special-purpose hardware stores call addresses for function calls and returns. For each call, the hardware stores a record of whether successive branches are taken or not taken. For example, the first row specifies a function call residing at address "X" followed by a taken branch, a not taken branch and then three taken branches. The second row specifies a function call residing at address "Y" and then no branches. The third row specifies a return from the function call to address "Y+4".

Finally, the fourth row specifies a function call residing at address "Z" followed by a two taken branches and a not taken branch.

Sample Reduced Executable Code

FIG. 9 illustrates a sample portion of reduced executable code 132 including prefetch instructions in accordance with an embodiment of the present invention.

The left-hand-side of FIG. 9 illustrates the structure of a section of executable code 130. This section is divided into a number of blocks 902, 904, 906, 908, 910 and 912, each of which is 64-bytes long and can be prefetched in a single prefetch operation. Note that executable code 130 first executes block 902, which performs a branch to either block 904 or block 906. Both blocks 904 and 906 perform a function call to block 908. Block 908 then performs a branch to either block 910 or block 912.

The right-hand-side of FIG. 9 illustrates the corresponding structure of a portion of reduced executable code 132. Reduced executable code 132 first prefetches blocks 902, 904 and 906. Next, the variable "addr" is loaded with the target address of the function call, and then reduced executable code 132 prefetches blocks 908, 910 and 912.

Note that reduced executable code 132 prefetches down both sides of the branches in executable code 130. In another embodiment of the present invention, reduced executable code 132 prefetches only down a predicted branch path in order to eliminate unnecessary instruction loads and instruction cache pollution.

Also note that the prefetch operation moves blocks of executable code 130 into L2 cache 113 in FIG. 1. Alternatively, the prefetch operation can cause instructions to be prefetched all the way into instruction cache 112 for primary processor 102.

The foregoing descriptions of embodiments of the invention have been presented for purposes of illustration and description only. They are not intended to be exhaustive or to limit the invention to the forms disclosed. Accordingly, many modifications and variations will be apparent to practitioners skilled in the art. Additionally, the above disclosure is not intended to limit the invention. The scope of the invention is defined by the appended claims.

What Is Claimed Is:

1. A method for prefetching instructions by using an assist processor to perform prefetch operations in advance of a primary processor, comprising:

executing executable code on the primary processor;

simultaneously executing a reduced version of the executable code on the assist processor, wherein the reduced version of the executable code executes more quickly than the executable code, and performs prefetch operations for the primary processor in advance of when the primary processor requires the instructions; and

storing the prefetched instructions into a cache that is accessible by the primary processor so that the primary processor is able to access the prefetched instructions without having to retrieve the prefetched instructions from a main memory.

2. The method of claim 1, further comprising, prior to executing the executable code:

compiling source code into executable code for the primary processor;

profiling the executable code to create instruction traces for frequently referenced portions of the executable code;

producing the reduced version of the executable code for the assist processor by producing prefetch instructions to prefetch portions of the instruction traces into the cache that is accessible by the primary processor; and

inserting communication instructions into the executable code for the primary processor and into the reduced version of the executable code for the assist processor to transfer progress information from the primary processor to the assist processor;

wherein the progress information triggers the assist processor to perform the prefetch operations.

3. The method of claim 2, wherein the process of compiling the source code and the process of producing the reduced version of the executable code are carried out by a compiler.

4. The method of claim 2, wherein if the progress information indicates to the assist processor that the assist processor has prefetched instructions down the wrong path, the reduced version of the executable code causes the assist processor to discontinue prefetching.

5. The method of claim 1, wherein the reduced version of the executable code is configured to:

read control flow history information from special-purpose hardware that records branch history information and call history information;

construct a predicted path through the executable code based on the control flow history information; and to

perform prefetch operations down the predicted path in order to prefetch instructions for the primary processor.

6. The method of claim 1, wherein producing the reduced version of the executable code involves:

constructing a control flow graph for the executable code;

removing loops from the control flow graph;

removing executable code instructions unrelated to the control flow graph; and

inserting the prefetch instructions into the reduced version of the executable code to prefetch instructions from the executable code for the primary processor.

7. The method of claim 1, wherein performing the prefetch operations involves prefetching cache blocks containing multiple instructions for the primary processor.

8. The method of claim 1, further comprising periodically sending the progress information from the primary processor to the assist processor through a one-way communication channel.

9. A computer-readable storage medium storing instructions that when executed by a computer cause the computer to perform a method for prefetching instructions by using an assist processor to perform prefetch operations in advance of a primary processor, comprising:

executing executable code on the primary processor;

simultaneously executing a reduced version of the executable code on the assist processor, wherein the reduced version of the executable code executes more quickly than the executable code, and performs prefetch operations for the primary processor in advance of when the primary processor requires the instructions; and

storing the prefetched instructions into a cache that is accessible by the primary processor so that the primary processor is able to access the prefetched instructions without having to retrieve the prefetched instructions from a main memory.

10. The computer-readable storage medium of claim 9, wherein prior to executing the executable code, the method further comprises:

compiling source code into executable code for the primary processor;

profiling the executable code to create instruction traces for frequently referenced portions of the executable code;

producing the reduced version of the executable code for the assist processor by producing prefetch instructions to prefetch portions of the instruction traces into the cache that is accessible by the primary processor; and

inserting communication instructions into the executable code for the primary processor and into the reduced version of the executable code for the assist processor to transfer progress information from the primary processor to the assist processor;

wherein the progress information triggers the assist processor to perform the prefetch operations.

11. The computer-readable storage medium of claim 10, wherein the process of compiling the source code and the process of producing the reduced version of the executable code are carried out by a compiler.

12. The computer-readable storage medium of claim 10, wherein if the progress information indicates to the assist processor that the assist processor has prefetched instructions down the wrong path, the reduced version of the executable code causes the assist processor to discontinue prefetching.

13. The computer-readable storage medium of claim 9, wherein the reduced version of the executable code is configured to:

read control flow history information from special-purpose hardware that records branch history information and call history information;

construct a predicted path through the executable code based on the control flow history information; and to

perform prefetch operations down the predicted path in order to prefetch instructions for the primary processor.

14. The computer-readable storage medium of claim 9, wherein producing the reduced version of the executable code involves:

constructing a control flow graph for the executable code;

removing loops from the control flow graph;

removing executable code instructions unrelated to the control flow graph; and

inserting the prefetch instructions into the reduced version of the executable code to prefetch instructions from the executable code for the primary processor.

15. The computer-readable storage medium of claim 9, wherein performing the prefetch operations involves prefetching cache blocks containing multiple instructions for the primary processor.

16. The computer-readable storage medium of claim 9, wherein the method further comprises periodically sending the progress information from the primary processor to the assist processor through a one-way communication channel.

17. An apparatus that facilitates prefetching from memory, comprising:

a primary processor that is configured to execute executable code;

an assist processor that is configured to simultaneously execute a reduced version of the executable code, wherein the reduced version of the executable code executes more quickly than the executable code, and performs prefetch operations for the primary processor in advance of when the primary processor requires the instructions; and

a cache that is accessible by the primary processor and is configured to store the prefetched instructions so that the primary processor is able to access the prefetched instructions without having to retrieve the prefetched instructions from a main memory.

18. The apparatus of claim 17, further comprising a compilation mechanism that is configured to:

compile source code into executable code for the primary processor;

profile the executable code to create instruction traces for frequently referenced portions of the executable code;

produce the reduced version of the executable code for the assist processor by producing prefetch instructions to prefetch portions of the instruction traces into the cache that is accessible by the primary processor; and to

insert communication instructions into the executable code for the primary processor and into the reduced version of the executable code for the assist processor to transfer progress information from the primary processor to the assist processor;

wherein the progress information triggers the assist processor to perform the prefetch operations.

19. The apparatus of claim 18, wherein if the progress information indicates to the assist processor that the assist processor has prefetched instructions down the wrong path, the reduced version of the executable code causes the assist processor to discontinue prefetching.

20. The apparatus of claim 17, further comprising special-purpose hardware that records branch history information and call history information for the executable code, and wherein the reduced version of the executable code is configured to:

read control flow history information from special-purpose hardware that records branch history information and call history information;

construct a predicted path through the executable code based on the control flow history information; and to

perform prefetch operations down the predicted path in order to prefetch instructions for the primary processor.

21. The apparatus of claim 17, wherein in producing the reduced version of the executable code, the compilation mechanism is configured to:

construct a control flow graph for the executable code;

remove loops from the control flow graph;

remove executable code instructions unrelated to the control flow graph; and to

insert the prefetch instructions into the reduced version of the executable code to prefetch instructions from the executable code for the primary processor.

22. The apparatus of claim 17, wherein the prefetch operations are configured to prefetch cache blocks containing multiple instructions for the primary processor.

23. The apparatus of claim 17, wherein the primary processor and the assist processor reside on the same semiconductor chip.

24. The apparatus of claim 17, wherein the primary processor and the assist processor reside on distinct semiconductor chips.

25. The apparatus of claim 17, wherein the assist processor is a simplified version of the primary processor without hardware that is unrelated to memory access operations.

5

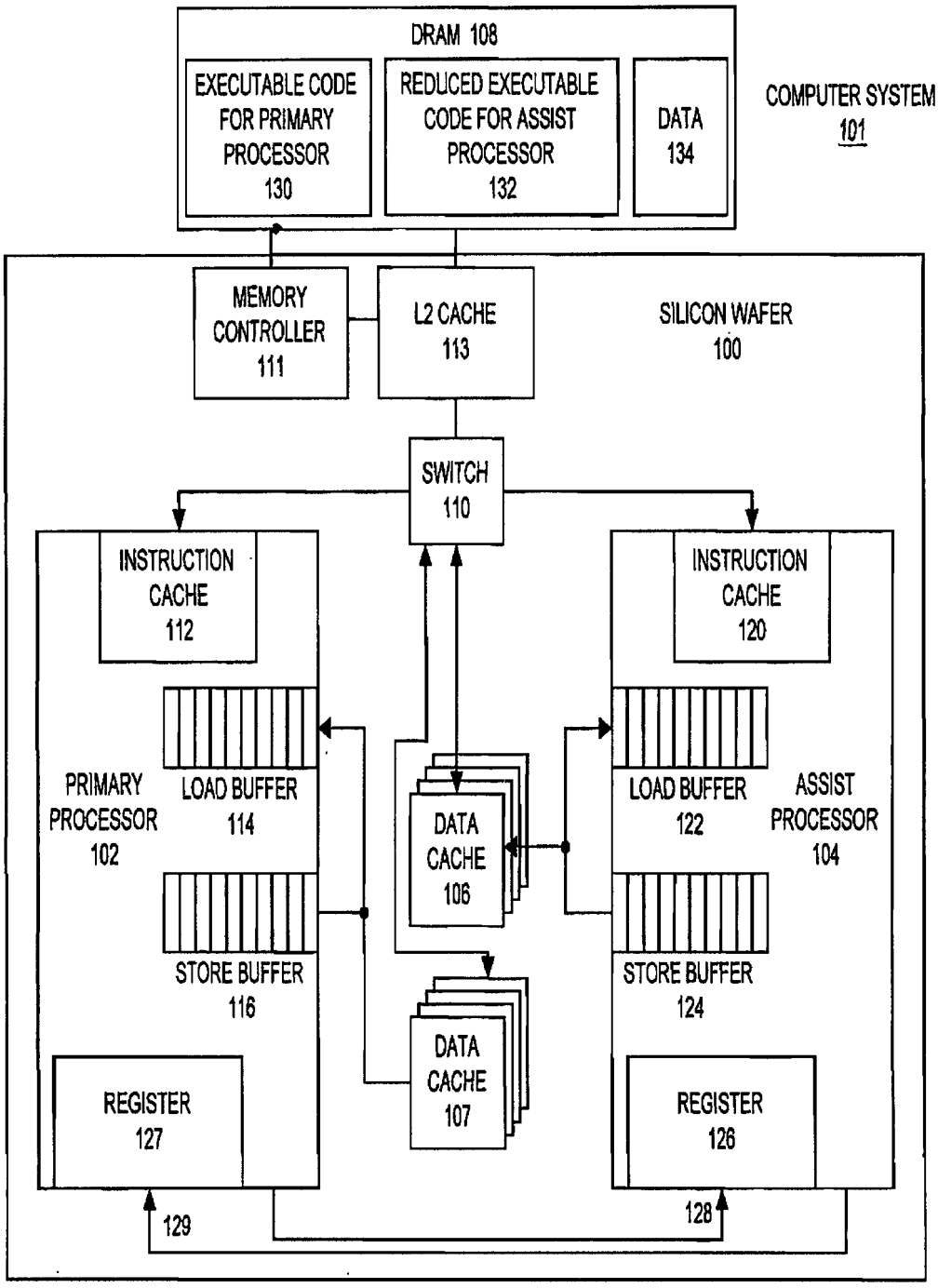

DRAM 108
EXECUTABLE CODE FOR PRIMARY PROCESSOR 130
REDUCED EXECUTABLE CODE FOR ASSIST PROCESSOR 132
DATA 134
COMPUTER SYSTEM 101
MEMORY CONTROLLER 111
L2 CACHE 113
SILICON WAFER 100
SWITCH 110
INSTRUCTION CACHE 112
INSTRUCTION CACHE 120
PRIMARY PROCESSOR 102
LOAD BUFFER 114
LOAD BUFFER 122
ASSIST PROCESSOR 104
DATA CACHE 106
STORE BUFFER 116
STORE BUFFER 124
DATA CACHE 107
REGISTER 127
REGISTER 126
129
128

FIG. 1

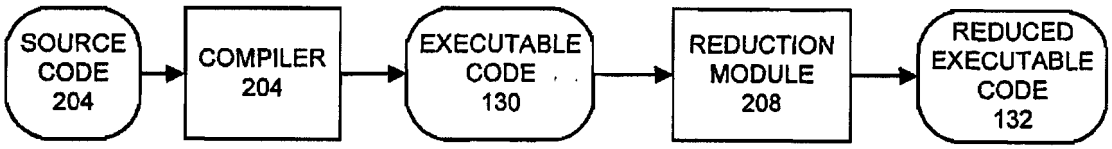

SOURCE CODE 204
COMPILER 204
EXECUTABLE CODE 130
REDUCTION MODULE 208
REDUCED EXECUTABLE CODE 132

FIG. 2

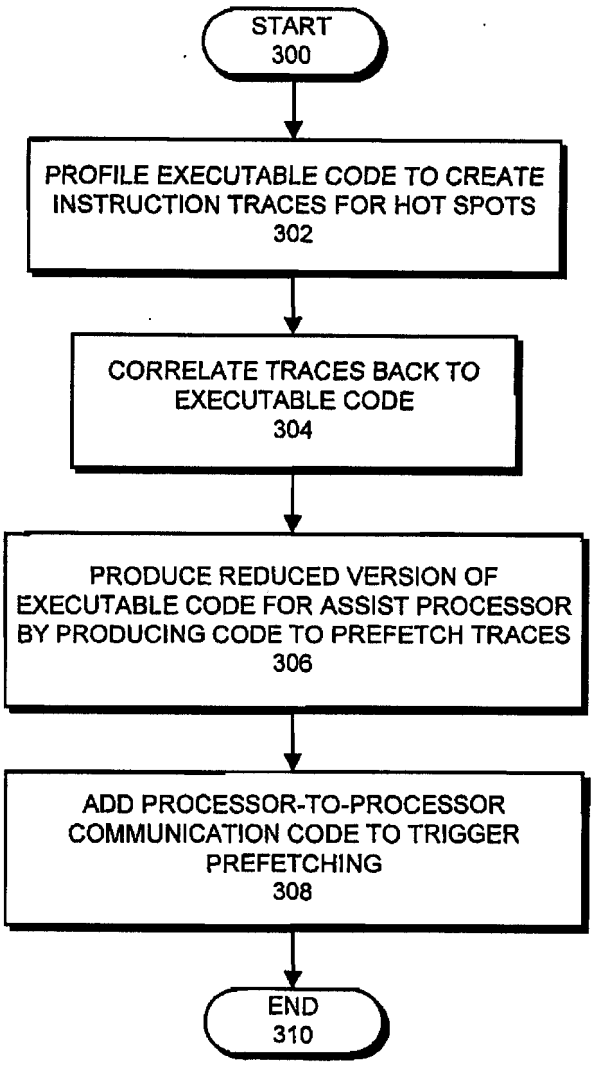

START 300
PROFILE EXECUTABLE CODE TO CREATE INSTRUCTION TRACES FOR HOT SPOTS 302
CORRELATE TRACES BACK TO EXECUTABLE CODE 304
PRODUCE REDUCED VERSION OF EXECUTABLE CODE FOR ASSIST PROCESSOR BY PRODUCING CODE TO PREFETCH TRACES 306
ADD PROCESSOR-TO-PROCESSOR COMMUNICATION CODE TO TRIGGER PREFETCHING 308
END 310

FIG. 3

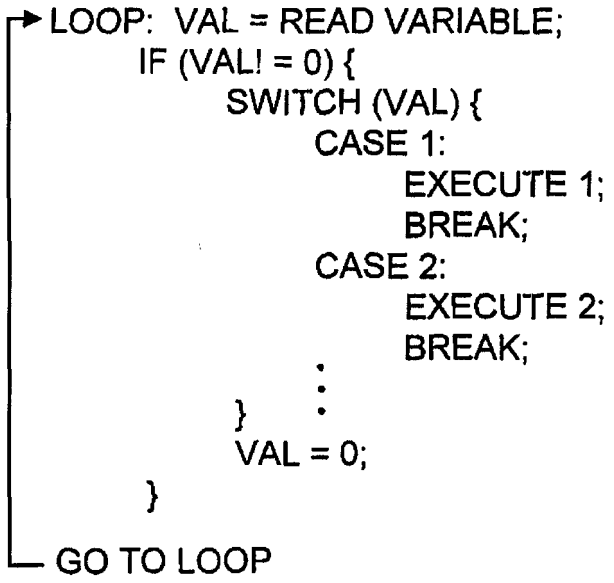

LOOP: VAL = READ VARIABLE;
IF (VAL! = 0) {
SWITCH (VAL) {
CASE 1:
EXECUTE 1;
BREAK;
CASE 2:
EXECUTE 2;
BREAK;
}
VAL = 0;
}
GO TO LOOP

FIG. 4

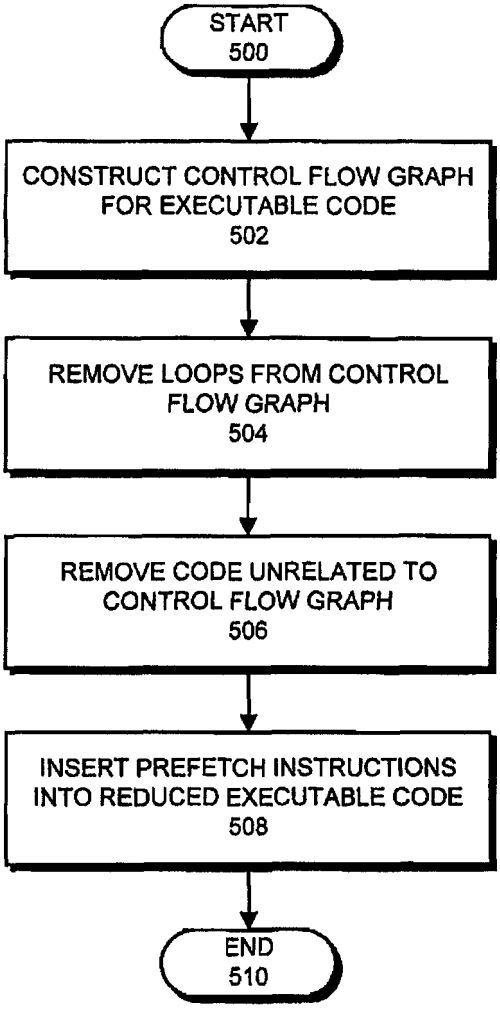

START
500
CONSTRUCT CONTROL FLOW GRAPH FOR EXECUTABLE CODE
502
REMOVE LOOPS FROM CONTROL FLOW GRAPH
504
REMOVE CODE UNRELATED TO CONTROL FLOW GRAPH
506
INSERT PREFETCH INSTRUCTIONS INTO REDUCED EXECUTABLE CODE
508
END
510

FIG. 5

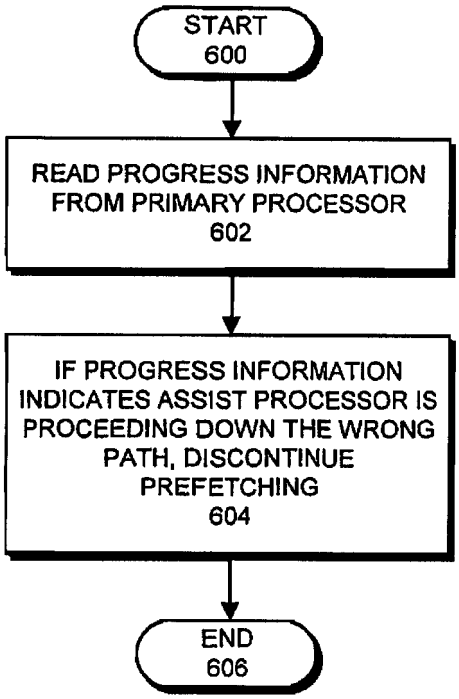


FIG. 6

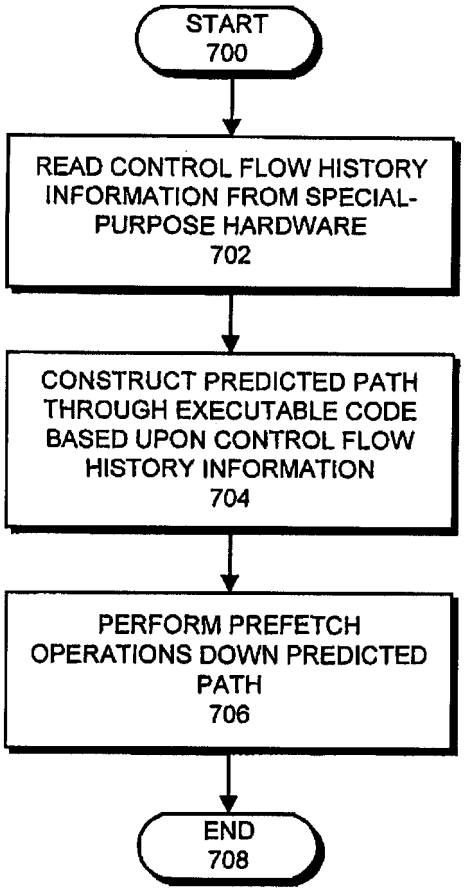


FIG. 7

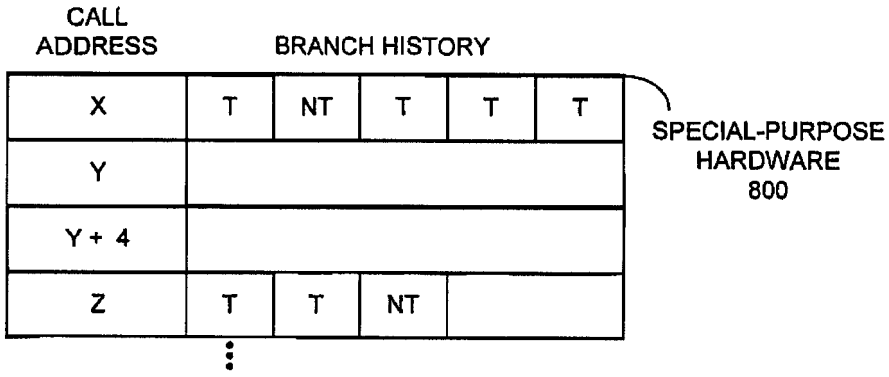
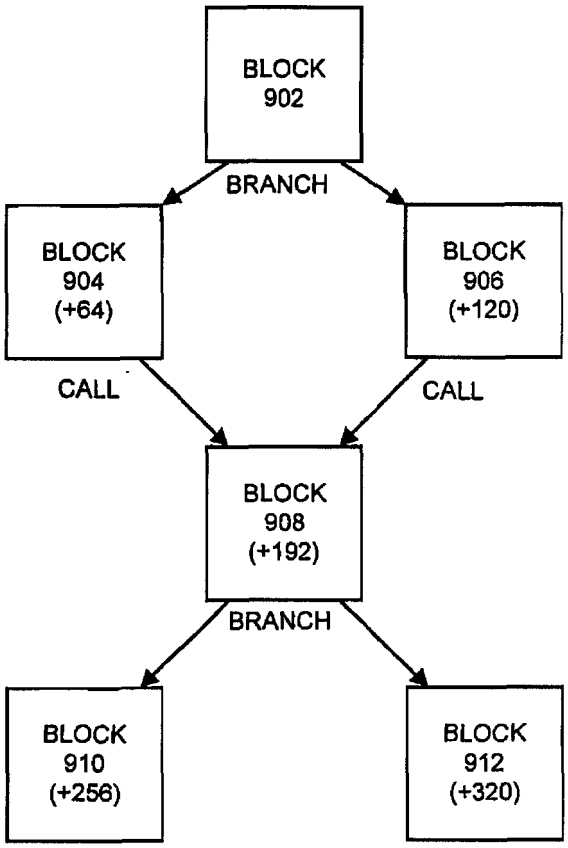


FIG. 8



PREFETCH ADDR;
ADDR = ADDR + 64;
PREFETCH ADDR;
ADDR = ADDR + 64;
PREFETCH ADDR;
ADDR = ADDRESS OF CALL;
PREFETCH ADDR;
ADDR = ADDR + 64;
PREFETCH ADDR;
ADDR = ADDR + 64;
PREFETCH ADDR;

FIG. 9

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
14 March 2002 (14.03.2002)

PCT

(10) International Publication Number
WO 02/21268 A3

(51) International Patent Classification[7]: G06F 9/38

(21) International Application Number: PCT/US01/41962

(22) International Filing Date: 30 August 2001 (30.08.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/231,452 8 September 2000 (08.09.2000) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).

(72) Inventors: CHAUDHRY, Shailender; 1200 Gough St., Apt. 10F, San Francisco, CA 94109 (US). TREMBLAY, Marc; 140 Hanna Way, Menlo Park, CA 94025 (US).

(74) Agent: PARK, Richard; 508 2nd Street, Suite 201, Davis, CA 95616 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR USING AN ASSIST PROCESSOR TO PREFETCH INSTRUCTIONS FOR A PRIMARY PROCESSOR

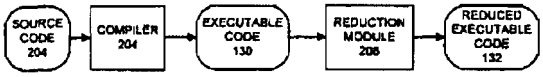
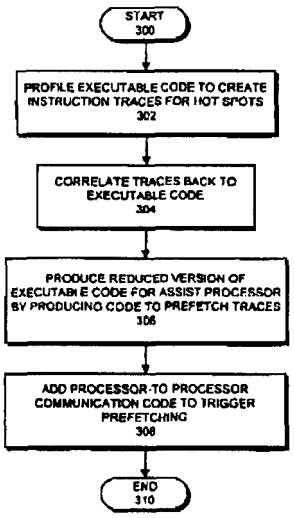


FIG. 2



(57) Abstract: One embodiment of the present invention provides a system that prefetches instructions by using an assist processor to perform prefetch operations in advance of a primary processor. The system operates by executing executable code on the primary processor, and simultaneously executing a reduced version of the executable code on the assist processor. This reduced version of the executable code executes more quickly than the executable code, and performs prefetch operations for the primary processor in advance of when the primary processor requires the instructions. The system also stores the prefetched instructions into a cache that is accessible by the primary processor so that the primary processor is able to access the prefetched instructions without having to retrieve the prefetched instructions from a main memory. In one embodiment of the present invention, prior to executing the executable code, the system compiles source code into executable code for the primary processor. Next, the system profiles the executable code to create instruction traces for frequently referenced portions of the executable code. The system then produces the reduced version of the executable code for the assist processor by producing prefetch instructions to prefetch portions of the instruction traces into a cache that is accessible by the primary processor. The system also inserts communication instructions into the executable code for the primary processor and into the reduced version of the executable code for the assist processor to transfer progress information from the primary processor to the assist processor. This progress information triggers the assist processor to perform the prefetch operations.

WO 02/21268 A3

(88) Date of publication of the international search report:
6 June 2002

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/41962

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/38

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 00 38050 A (INTEL CORP) 29 June 2000 (2000-06-29) the whole document ---	1-3,7, 9-11,15, 17,18,22
A	US 5 961 631 A (DEVEREUX IAN VICTOR ET AL) 5 October 1999 (1999-10-05) abstract column 5, line 50 -column 6, line 20 column 8, line 10 -column 10, line 44 --- -/--	1,8,17

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
3 April 2002	11/04/2002

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Moraiti, M

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/41962

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	SAKALAY F E: "STORAGE HIERARCHY CONTROL SYSTEM" IBM TECHNICAL DISCLOSURE BULLETIN, IBM CORP. NEW YORK, US, vol. 15, no. 4, 1 September 1972 (1972-09-01), pages 1100-1101, XP002002415 ISSN: 0018-8689 the whole document -----	1,9,17

1

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Ir ational Application No
PCT/US 01/41962

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 0038050 A	29-06-2000	US 6205544 B1	20-03-2001
		AU 2189800 A	12-07-2000
		WO 0038050 A1	29-06-2000
US 5961631 A	05-10-1999	NONE	